Attorney Docket: 0257061C/2631C

CERTIFICATE OF MAIL

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail in an envelope addressed to Mail Stop Amendment, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on **June 18, 2004**.

Jinhy Nguyen

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re Application of: Date: June 18, 2004

Lav IVANOVIC et al. Confirmation No.: To Be Assigned

Serial No: 10/829,408 Group Art Unit: To Be Assigned

Filed: April 20, 2004 Examiner: To Be Assigned

For: AUTOMATIC CALIBRATION OF A MASKING PROCESS SIMULATOR

Mail Stop Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION UNDER 37C.F.R. 1.131

We, **Lav IVANOVIC, Paul FILSETH, and Mario GARZA**, hereby declare that:

1. We are inventors of the subject matter recited in the claims of the above-identified application.

2. Prior to October 9, 2001, we conceived of the idea of an automatic method to calibrate a masking process simulator that analyzes the difference between the aerial image produced by a simulator and the actual pattern produced by the masking process in order to improve the

results of the simulator, as described and claimed in our application.

3. We conceived of the invention and reduced it to practice while working for LSI Logic Corporation, having a principal place of business in Milpitas, California, in the OPC group, which includes LSI Logic personnel from California as well as from LSI Logic. International located in Russia.

4. Attached Exhibit A is a copy of a document transmitted by facsimile from LSI logic International to LSI logic (CA) on January 30, 2001, describing the "Montenegro project" (see pg. 3). Page 4 is a flow diagram of the project, and the blocks shown therein are described on page 5-8 (see, for example, block B12 and accompanying text).

5. Attached Exhibit B is a screenshot of a directory structure stored on LSI logic computers showing "C" files in pre-build state that were used implement the present invention. As shown, all the files have dates of completion prior to February 29, 2000.

6. Attached Exhibit C is a source code listing of the relevant portions of the source files for implementing the claimed operations above.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are

punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Lav IVANOVIC — June, 17, 2004
Date

Paul FILSETH — June 10, 2004
Date

Mario GARZA — June 17/04
Date

Проект «Монтенегро»

С.В.Алешин Отчет за январь 2001

За прошедший период группа проанализировала состояние OPC-проекта,наметила пути решения и получила конкретные результаты в некоторых направлениях.

Было определено,что дальнейшее продвижение связано с решением задач в трех главных проблемах.

1. Необходимо дальнейшее развитие и совершенствование программы «Молотов», то есть программы редактирования дизайнов и симуляции масок. Ряд недостатков действующей версии выявился при обработке больших дизайнов,все они сообщены Полу для исправления.Требования к «Молотову» возрастают и в связи с тем , что эта программа должна стать частью единого программного комплекса OPC.

2.Необходимо дальнейшее развитие программы OPC ,чтобы обеспечить более быструю обработку больших дизайнов и, с другой стороны, еще более высокое качество обработки малых плотных дизайнов (памяти). Одновременно необходимо вести разработку пользовательского интерфейса.

3.Необходимо приступить к рассмотрению проблемы автоматизации процесса калибровки симулятора в зависимости от свойств резиста и оптической системы.

Мы расчитываем,что Пол обеспечит реализацию п.1, даст нам описание п.3 (калибровки) на достаточно высоком уровне формализации,а также примет участие в обсуждении и реализации пользовательского интерфейса.

В плане реализации п.2 были получены следующие результаты.

Е.Егоров главное внимание уделил блоку выходной информации. Ставилась задача получить максимально компактное описание результирующей маски большого дизайна. Для этого разработан специальный формат,названный

EXHIBIT A

ОРС-форматом,который позволяет получать описания,объем которых на порядок меньше тех которые использовались до сих пор.

Увеличение скорости обработки больших дизайнов возможно на пути более активного использования синтаксических процедур и выделения в дизайне структур специального вида - примером таких структур могут быть надписи, буквы, периодические структуры,структуры допускающие одномерную обработку, оптически независимые подструктуры и т.д.

Один из первых (начальных) результатов в этом направлении получил С.Родин - процедура выделения ячеек типа «буква».

М.Медведева реализовала процедуру,позволяющую получать границу изображения не совпадающую с границей исходного дизайна,а с некоторым,заданным сдвигом.Это дает возможность реализовать идею Дженсена о «расширении» исходного дизайна.

Г.Белокопытов провел серию экспериментов по вычислению двухслойных фазовых масок,вычисляемых на основе взаимодействия двух слоев чипа (Poli и Island). Идея таких масок была предложена Дженсеном. Оказалось, что такие маски действительно повышают крутизну профиля интенсивности в районе границы ,а следовательно могут улучшить результат фотолитографии.В то же время в плотных дизайнах со сложными конфигурации могут возникать проблемы нехватки свободного пространства.

Н.Анисимов предложил общую схему интерфейса пользователя ОРС-комплекса.Эта схема направлена Полу для обсуждения.

А.Чернушич приступил к изучению кода программы ОРС. Перед ним поставлена задача предложить поправки,которые улучшат тестируемость программы а также ликвидируют имеющиеся сейчас эффекты «утечки памяти».

LSI Logic International

MONTENEGRO project **Месячный отчет**

Исполнитель: Н.Ф.Анисимов **Период: январь 2001 г.**

Основные результаты.

1 Разработка алгоритмов.

- разработана общая блок-схема интерфейса пользователя для решения задачи коррекции;
- описано функционирование отдельных блоков данной схемы;

Блок-схема и описание функционирования ее блоков прилагается.

2 Планы на следующий месяц.

2.1 Разработка алгоритмов.

- продолжение разработки интерфейса пользователя;
- программирование отдельных блоков интерфейса;

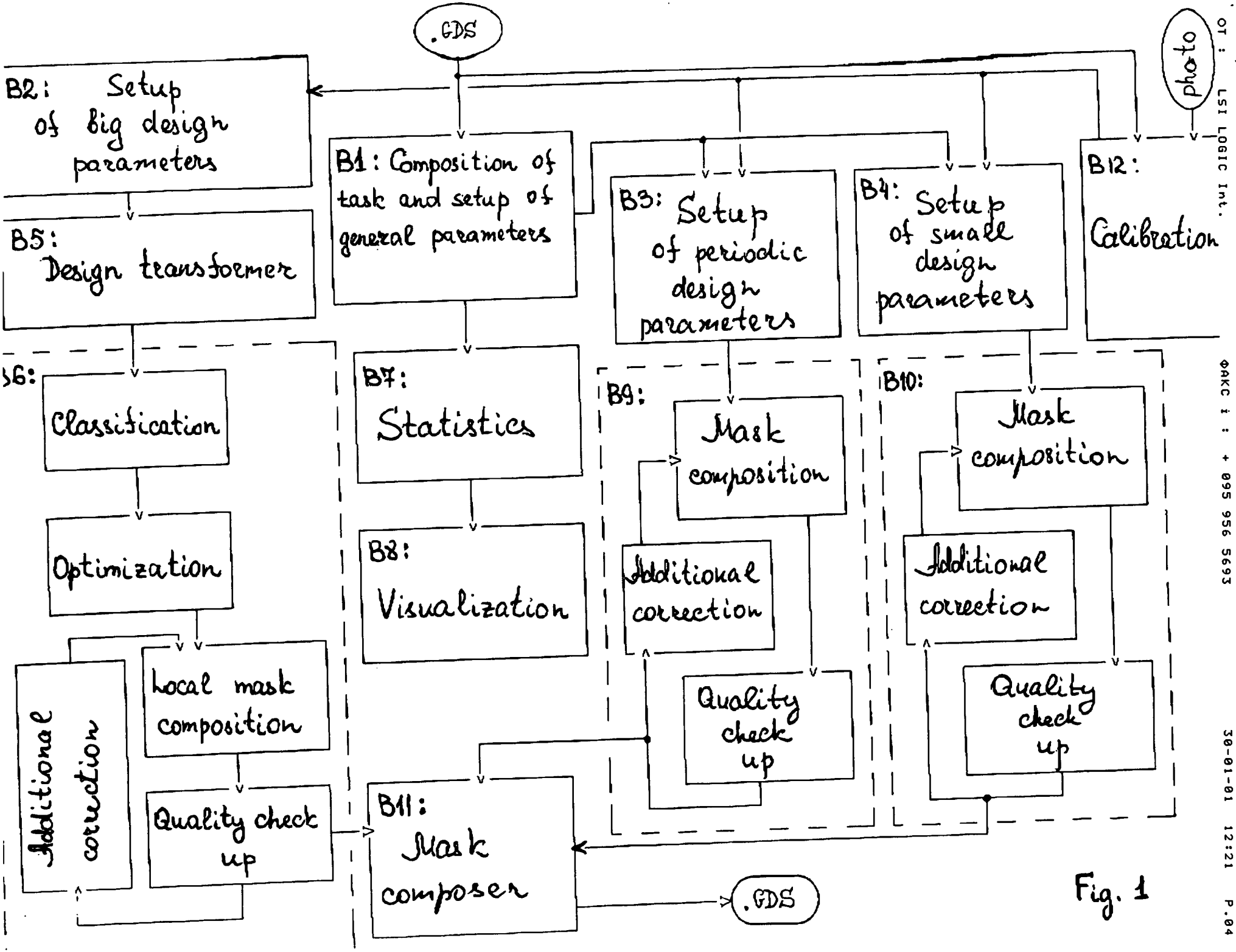

Fig. 1

1 Flowchart of user interface and description of its links and function.

General flowchart of user interface is represented at fig.1.
Description of its action is given below.

1.1 Block B1: composition of task and setup of general parameters.

- determination of paths of search for
  - input data;
  - parameter files;
  - output data.
- determination of general parameters of design:
  - input design gds-file name;
  - top structure name in input gds-file;
  - format of output data;
  - output file name;
  - description of layers for input gds-file;
  - description of layers for output mask file;
  - types of mask used.
- choice of models and their parameters setup:
  - optical model;
  - simulator model;
  - resist and exposure model;
  - etching model.
- determination of the type of design problem:
  - simple design;
  - complex design.

1.2 Block B2 : setup of big design parameters.

Setup of the specific parameters for processing of big design, such as:

- classification algorithm parameters;
- optimization algorithm parameters;
- mask composition algorithm parameters;
- parameters of quality check and additional correction.

1.3 Block B3 : setup of periodic design parameters.

Setup of the specific parameters for processing of periodic design, such as:

- classification algorithm parameters;
- optimization algorithm parameters;
- mask composition algorithm parameters;
- parameters of quality check and additional correction.

1.4 Block B4: setup of small design parameters.

Setup of the specific paramcters for processing of small design, such as:

- classification algorithm parameters;
- optimization algorithm parameters;
- mask composition algorithm parameters;
- paramcters of quality check and additional correction.

1.5 Block B5: design transformer.

Completes decomposition of initial complex design on to partial fragments, which have move simple structure (periodic, memory, text, secluded, etc). Generation of the data flow for further processing of fragments and formation of the task stack.

Design fragmentation may be completed:

- by cell type or list of cell types;
- by cell name or list of cell names;
- by chosen region of list of regions (or with exclusion of some regions);
- by density chart.

1.6 Block B6: classification, optimization, local mask composition, quality check up and additional correction for big design.

This block contains basic OPC procedures modified for complex designs. The more detalled description of these procedures will be added in nearest future.

1.7 Block B7: statistics.

This block will contain a set of procedures, which enable to user:

- to obtain numerical and qualitative estimation of the design (volume, complexity etc);
- to display and analyze the sequense of stages of design processing;
- to create protocols of design correction;
- to search the information current status of processes and on resourses in use;
- to create different log-files.

1.8 Block B8: visualization.

The rocedures of this block enable user to execute such operation as:

- display information given by block B7 "Statistics";
- to run visualization tools, such as Molotof or Artwork Editor, for scan and edition of input designs, masks, simulation results, etc;
- to build hystograms and graph which illustrate results of the design processing;
- to convert input data represented in different graphics formats(gds, bgl, tgl, MEBES, CIF, etc).

1.9 Block B9: mask composition, quality check up and additional correction for periodic design.

This block contains basic OPC procedures modified for periodic designs. The more detalled description of these procedures will be added in nearest future.

1.10 Block B10: mask composition, quality check up and additional correction for small design.

This block contains basic OPC procedures modified for small designs. The more detalled description of these procedures will be added in nearest future.

1.11 Block B11: mask composer.

This block executes the procedure of output mask composition for entire design from local and fragment masks and forms output gds-file.

1.12 Block B12: calibration.

This block will be intended to make (in automatic and/or interactive mode) the procedure of adjustment of models parameters to empirical data obtained for some test patterns. Such adjustment will give more adecuate description of phisical and technological processes (diffraction image formation, exposure, post-exposure development and mask making) for opc.

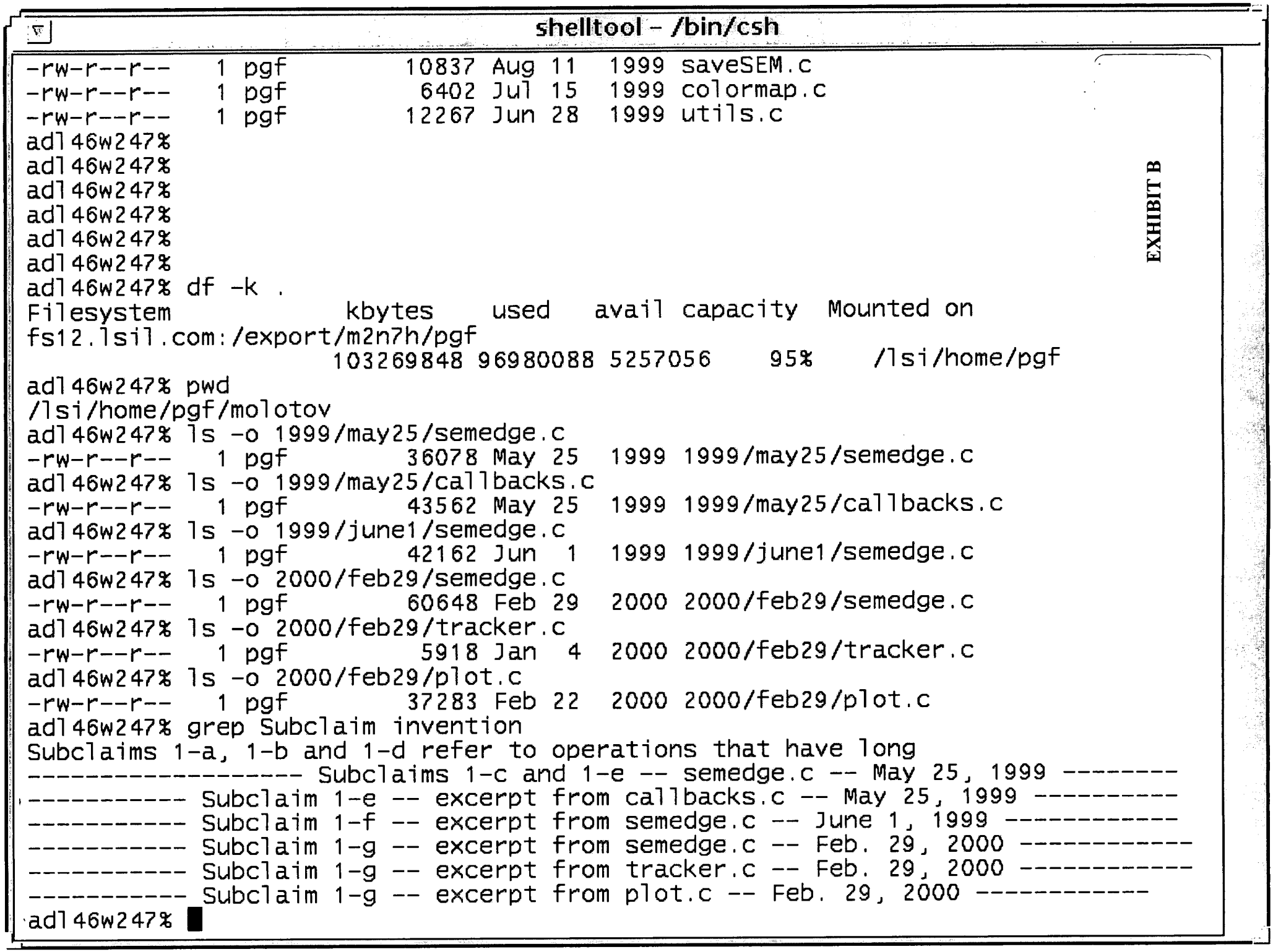
shelltool - /bin/csh
-rw-r--r-- 1 pgf 10837 Aug 11 1999 saveSEM.c
-rw-r--r-- 1 pgf   6402 Jul 15 1999 colormap.c
-rw-r--r-- 1 pgf 12267 Jun 28 1999 utils.c
adl46w247%
adl46w247%
adl46w247%
adl46w247%
adl46w247%
adl46w247%
adl46w247% df -k .
Filesystem kbytes used avail capacity Mounted on
fs12.lsil.com:/export/m2n7h/pgf
  103269848 96980088 5257056 95% /lsi/home/pgf
adl46w247% pwd
/lsi/home/pgf/molotov
adl46w247% ls -o 1999/may25/semedge.c
-rw-r--r-- 1 pgf 36078 May 25 1999 1999/may25/semedge.c
adl46w247% ls -o 1999/may25/callbacks.c
-rw-r--r-- 1 pgf 43562 May 25 1999 1999/may25/callbacks.c
adl46w247% ls -o 1999/june1/semedge.c
-rw-r--r-- 1 pgf 42162 Jun 1 1999 1999/june1/semedge.c
adl46w247% ls -o 2000/feb29/semedge.c
-rw-r--r-- 1 pgf 60648 Feb 29 2000 2000/feb29/semedge.c
adl46w247% ls -o 2000/feb29/tracker.c
-rw-r--r-- 1 pgf   5918 Jan 4 2000 2000/feb29/tracker.c
adl46w247% ls -o 2000/feb29/plot.c
-rw-r--r-- 1 pgf 37283 Feb 22 2000 2000/feb29/plot.c
adl46w247% grep Subclaim invention
Subclaims 1-a, 1-b and 1-d refer to operations that have long
---------- Subclaims 1-c and 1-e -- semedge.c -- May 25, 1999 --------
---------- Subclaim 1-e -- excerpt from callbacks.c -- May 25, 1999 ----------
---------- Subclaim 1-f -- excerpt from semedge.c -- June 1, 1999 ----------
---------- Subclaim 1-g -- excerpt from semedge.c -- Feb. 29, 2000 ----------
---------- Subclaim 1-g -- excerpt from tracker.c -- Feb. 29, 2000 ----------
---------- Subclaim 1-g -- excerpt from plot.c -- Feb. 29, 2000 ----------
adl46w247%

EXHIBIT B

```
------------------- Subclaims 1-c and 1-e -- semedge.c -- May 25, 1999
--------

    This file contains a computer-vision algorithm for detecting the
    edges of features on a digitized gray-scale image of a pattern,
    and a procedure for drawing the detected edges onto the digital
    image.

/*************************************************************************
*******
 * semedge.c
 *
 * Find the features in a SEM
 *
*************************************************************************
******/
#include "Screens.h"

#define ZOOM_STEPS 20
#define HASH_SIZE 4096

#define TOO_WIDE 100    // Limit on SEM feature edge width
#define HIST_SIZE 20    // HIST_SIZE > 2 * log2(TOO_WIDE)

#define NEAR(v, t, r) ((unsigned) ((v) - (t - r)) <= (r * 2)) #define
PIF if (cfp) fprintf(cfp,
#define ENDPIF );
#define ENDPIFF ), fflush(cfp);

typedef struct {
    char which_dir;
} connectionsrec, *connections;

typedef struct locREC {
    struct locREC *next;
    int color;
    short x, y;
    short feature_id;
    connectionsrec left, right;
    char cleft, cright;
    char reached;
} locrec, *loc;

typedef struct pairingREC {
    struct pairingREC *next;
    struct splotchREC *s1, *s2;
    int count;
} pairingrec, *pairing;

typedef struct splotchREC {
    struct splotchREC *next;
    struct splotchREC *mesa;
    int id;
    int dir;
    loc *edge;
    pairing partners;
    int polarity;
```

```
} splotchrec, *splotch;

/*************************** Variables
*******************************/

static int bright[256];
static int variation;
static int lo_x, hi_x, lo_y, hi_y;
static int curcolor = 0;
static int array_size = 0;
static int *profile, *a1, *a3, *a9, *rising, *falling;
static Display *theDisp;
// static FILE *cfp = (FILE *) -1;
static FILE *cfp = NULL;
static loc lochash[HASH_SIZE];
static int grad_radius, edge_width, box_size;
static int features[500];
static int histogram[HIST_SIZE + 1];
static int feature_id;


static connectionsrec empty_connections = {-1};

extern Colormap my_colormap;

/*************************** Functions
*******************************/

#define EOFF 3

draw_feature_edges_on_SEM(Widget w, graphics_data *data)
{
splotch s;
loc p;
int i, ix, iy, j, ox, oy, bad, obad, phase;
Display        *dpy = XtDisplay(w);
Window          win = XtWindow(w);
GC gc;
pmversion pm;
double factor;
connections c, r;
static string color[] = {"Violet", "Green", "Red", "Blue", "Yellow"};

    pm = data->curpm;
    factor = ((double) pm->zoom_level) / ZOOM_STEPS;
    gc = data->gc;
    gc = data->gcSEMLabel;
    for (j = 0; j < 5; ++j) {
      XSetForeground(dpy, gc, color_code(color[j], w));
      loop_over_list(s, data->features) {
          obad = ox = oy = -1;
          for (i = 0; s->edge[i] != NULL; ++i) {
            p = s->edge[i];
            ix = (int) (factor * p->x + 0.5);
            iy = (int) (factor * p->y + 0.5);
            if (data->flipX)
                ix = data->sem->xres - ix;
```

```
            if (data->flipY)
                iy = data->sem->yres - iy;
            /* Alternate color rules
            phase = s->mesa->id % 5;

            if (s->polarity >= 2)
                phase = 2;
            else if (s->polarity <= -2)
                phase = 3;
            else if (s->polarity == 0)
                phase = 4;
            else
                phase = 1;
            */
            if (s->polarity >= 2)
                phase = 4;
            else if (s->polarity <= -2)
                phase = 3;
            else
                phase = -1;
            if (phase == j) {
                // XDrawPoint(dpy, win, gc, ix + EOFF, iy + EOFF);
                if (i != 0)
                  XDrawLine(dpy, win, gc,
                        ix + EOFF, iy + EOFF, ox + EOFF, oy + EOFF);
            }
            ox = ix;
            oy = iy;
            obad = bad;
          }
      }
    }
}

static int brightness(graphics_data *data, int x, int y)
{
pmversion pm;
windo sem;
int c, pix;
XColor color;

    pm = data->curpm;
    sem = data->sem;
    if (x < 0)
      x = 0;
    if (x >= sem->xres)
      x = sem->xres;
    if (y < 0)
      y = 0;
    if (y >= sem->yres)
      y = sem->yres;
    c = pm->expImage->data[y * pm->width + x] & 255;
    if (bright[c] == 0) {
      pix = (int) XGetPixel(pm->expImage, x, y);
      color.pixel=pix;
      XQueryColor(theDisp,my_colormap,&color);
      bright[c] = (int)
```

```
            ((color.blue + color.green + color.red) * 0.333333 /
65.536);
      if (bright[c] == 0)
          bright[c] = 1;
    }
    return(bright[c]);
}

static expand_arrays(sem)
windo sem;
{
int needed;

    needed = (sem->xres > sem->yres ? sem->xres : sem->yres) + 25;
    if (needed <= array_size)
      return;
    array_size = needed;
    a1 = newarray(int, needed) + 11;
    profile = a1;
    a3 = newarray(int, needed) + 11;
    a9 = newarray(int, needed) + 11;
    rising = newarray(int, needed) + 11;
    falling = newarray(int, needed) + 11;
}

static profile_line(graphics_data *data, int start, int dx, int dy) {
int x, y, pix, i, j; XColor color; int val, last, diff; Boolean
falling; windo sem;

    sem = data->sem;
    expand_arrays(sem);
    if (dx == 0) {
      y = 0;
      x = start;
    } else if (dy == 0) {
      x = 0;
      y = start;
    } else if (dx == 1 && dy == 1) {
      if (start >= 0) {
          x = 0;
          y = start;
      } else {
          x = - start;
          y = 0;
      }
    } else if (dx == 1 && dy == -1) {
      if (start >= 0) {
          x = 0;
          y = sem->yres - 1 - start;
      } else {
          x = - start;
          y = sem->yres - 1;
      }
    } else {
      return;
    }
    last = 0;
```

```
    falling = True;
    for (i = 0; True; x += dx, y += dy, ++i) {
      if (x < 0 || x >= sem->xres)
          break;
      if (y < 0 || y >= sem->yres)
          break;
      val = brightness(data, x, y);
      profile[i] = val;
    }
    profile[i] = -1;
}

static int info_content(graphics_data *data,
            int start, int dx, int dy, Boolean calibrate)
{
int i, last, v, a, medium, total;
int low, high;
double f1, f2, f3;

    profile_line(data, start, dx, dy);
    medium = total = 0;
    low = high = 0;
    for (i = 3; profile[i] >= 0; ++i) {
      last = profile[i - 3];
      v = profile[i];
      a = v - last;
      if (a < 0)
          a = - a;
      if (a > 80)
          a = 0;
      total += a;
      if (v < 200)
          ++ low;
      if (v > 800)
          ++ high;
    }
    if (calibrate) {
      variation += total / i;
      return(0);
    }
    f1 = (double) high / i;
    f2 = (double) low / i;
    f3 = ((double) total / i) / variation;
    if (f1 > 0.9 && f3 < 0.5)
      i = 1;
    else if (f2 > 0.4 && f3 < 0.8)
      i = 2;
    else if (f1 + f2 > 0.9 && f3 < 0.6)
      i = 3;
    else
      i = 0;
    return(i);
}

static int find_border(graphics_data *data, int dx, int dir)
{
windo sem;
```

```
int out, in, mid, dy, range, last;

    sem = data->sem;
    dy = 1 - dx;
    range = dx ? sem->yres : sem->xres;
    if (dir == 1) {
      out = -1;
      in = range / 2;
    } else {
      out = range;
      in = range / 2;
    }
    last = -1;
    while (in != out + dir) {
      mid = (out + in) / 2;
      if (mid == last)
          printf("Screwup\n"), exit(1);
      if (info_content(data, mid, dx, dy, False) == 0)
          in = mid;
      else
          out = mid;
      last = mid;
    }
    return(in);
}

static identify_border_areas(graphics_data *data)
{
windo sem;

    sem = data->sem;
    variation = 0;
    info_content(data, (int) (sem->xres * 0.4), 0, 1, True);
    info_content(data, (int) (sem->xres * 0.5), 0, 1, True);
    info_content(data, (int) (sem->xres * 0.6), 0, 1, True);
    variation /= 3;
    lo_y = find_border(data, 1, 1);
    hi_y = find_border(data, 1, -1);
    lo_x = find_border(data, 0, 1);
    hi_x = find_border(data, 0, -1);
    PIF "Border: x = %d-%d, y = %d-%d\n\n", lo_x, hi_x, lo_y, hi_y
ENDPIFF }

static void record_pair_stats(int width)
{
int ix, pow;

    if (width <= 3) {
      ix = width;
    } else {
      for (pow = 2; (1 << pow) <= width; ++pow);
      ix = (width >> (pow - 2)) + pow * 2 - 4;
    }
    PIF "Width = %d, ix = %d\n", width, ix ENDPIF
    ++ histogram[ix];
}
```

```
static int ix_to_n(int i)
{
    return(((2 + (i & 1)) << (i >> 1)) >> 1);
}

static void analyze_pair_statistics()
{
int i, v, max, ix;

    ix = max = 0;
    for (i = 0; i < HIST_SIZE; ++i) {
      if (histogram[i] != 0)
          PIF "%d: range = %d to %d, num of pairs = %d\n", i,
                  ix_to_n(i), ix_to_n(i + 1) - 1,
                  histogram[i] ENDPIF
      v = (histogram[i] + histogram[i + 1]) * i;
      if (v >= max) {
          max = v;
          ix = i;
      }
    }
    edge_width = ix_to_n(ix + 1);
    PIF "Typical width of suspected edges = %d pixels\n", edge_width
ENDPIF
    printf("Typical width of suspected edges = %d pixels\n",
edge_width);
    grad_radius = edge_width / 4;
    if (grad_radius < 1)
      grad_radius = 1;
    if (grad_radius > 5)
      grad_radius = 5;
    // box_size = edge_width * 5 / 2;
    box_size = edge_width * 2;
}


static int find_features(int lo, int hi)
{
int i, e, last, base, j, min, count;

    j = (a1[lo] + a1[lo + 1]) >> 1;
    for (i = lo - 10; i < lo; ++i)
      a1[i] = j;
    j = (a1[hi] + a1[hi - 1]) >> 1;
    for (i = hi + 10; i > hi; --i)
      a1[i] = j;

    for (i = lo - 9; i <= hi + 9; ++i)
      a3[i] = (a1[i - 1] + a1[i] + a1[i + 1]) / 3;
    a3[lo - 10] = a3[lo - 9];
    a3[hi + 10] = a3[hi + 9];

    for (i = lo - 7; i <= hi + 7; ++i)
      a9[i] = (a3[i - 3] + a3[i] + a3[i + 3]) / 3;

    for (i = lo - 2; i <= hi + 2; ++i) {
      rising[i] = profile[i] - profile[i - 1] + a3[i + 1] - a3[i - 1] +
```

```
            a3[i + 1] - a9[i - 5];
      falling[i] = profile[i] - profile[i + 1] + a3[i - 1] - a3[i + 1]
+
            a3[i - 1] - a9[i + 5];
    }

    count = 0;
    last = -99;
    for (i = lo; i <= hi; ++i) {
      e = rising[i];
      if (e > 200 && e > rising[i - 1] && e >= rising[i + 1] &&
                  e > rising[i - 2] && e >= rising[i + 2]) {
          // fprintf(cfp, "  r[%d] = %d\n", i, e);
          last = i;
      }
      e = falling[i];
      if (e > 200 && e > falling[i - 1] && e >= falling[i + 1] &&
                  e > falling[i - 2] && e >= falling[i + 2]) {
          // fprintf(cfp, "    f[%d] = %d\n", i, -e);
          if (last != -99) {
            base = a3[last - 2];
            if (base < a3[i + 2])
                base = a3[i + 2];
            if (i <= last + 1) {
                min = profile[profile[i] < profile[last] ? i : last];
            } else {
                min = a3[last + 1];
                for (j = last + 2; j < i; ++j)
                  if (a3[j] < min)
                      min = a3[j];
            }
            PIF "Pair at %d-%d, edges = %d, height = %d\n",
                        last, i,
                        rising[last] + falling[i],
                        min - base ENDPIF
            if (i - last > TOO_WIDE) {
                PIF "    Rejected: too wide\n" ENDPIF
            } else if (min - base <= 0) {
                PIF "    Rejected: too low\n" ENDPIF
            } else {
                record_pair_stats(i - last);
                features[count++] = last;
                features[count++] = i;
                features[count] = -99;
            }

            last = -99;
          }
      }
    }
    if (cfp) fflush(cfp);
    return(count);
}

static int *cross_section(graphics_data *data, int start, int dx) { int
dy; int i, x, y, n, dir; loc p; int *ptr;
```

```
    dy = 1 - dx;
    if (cfp) fprintf(cfp, "cross_section %d, dx=%d, dy=%d\n", start,
dx, dy);
    profile_line(data, start, dx, dy);
    if (dx == 1)
      n = find_features(lo_x, hi_x);
    else
      n = find_features(lo_y, hi_y);
    ptr = newarray(int, n * 2 + 1);
    for (i = 0; i < n; ++i) {
      if (dx == 1) {
          x = features[i];
          y = start;
      } else {
          x = start;
          y = features[i];
      }
      ptr[2 * i] = x;
      ptr[2 * i + 1] = y;
    }
    ptr[2 * n] = -99;
    return(ptr);

}

static int octant, mag, level;
static int delta_x, delta_y;
static string dirname[] = {
      "ENE", "NNE", "NNW", "WNW", "WSW", "SSW", "SSE", "ESE" };

/* This approximately converts cartesian coordinates to polar
coordinates,
   based on pretending an octagon is a circle.  It returns the angle in
   45-degree chunks and leaves the radius in "mag". */
static int cartesian_to_polar(dx, dy)
int dx, dy;
{
static int octant_map[8] = {0, 1, 3, 2, 7, 6, 4, 5};
int adx, ady;
int ix;

    if (dx < 0)
      ix = 2, adx = - dx;
    else
      ix = 0, adx = dx;
    if (dy < 0)
      ix += 4, ady = - dy;
    else
      ady = dy;
    if (adx < ady)
      ++ ix, adx >>= 1;
    else
      ady >>= 1;
    mag = adx + ady;
    return(octant_map[ix]);
}
```

```
static splotch new_splotch(splotch *plis)
{
splotch s;

    s = new(splotchrec);
    s->id = feature_id;
    s->next = *plis;
    *plis = s;
    s->mesa = NULL;
    s->partners = NULL;
    s->edge = NULL;
    s->dir = 0;
    s->polarity = 0;
    return(s);
}

#define EDGE_EFFECT 5

static find_gradient(graphics_data *data, int x, int y)
{
static int octant_map[8] = {0, 1, 3, 2, 7, 6, 4, 5};
int i, j, k, dx, dy, adx, ady;
int val[9];

    for (k = 0, i = -grad_radius; i <= grad_radius; i += grad_radius)
      for (j = -grad_radius; j <= grad_radius; j += grad_radius)
          val[k++] = brightness(data, x + i, y + j);
    dx = (val[6] - val[0] + val[7] - val[1] + val[8] - val[2]) / 3;
    dy = (val[2] - val[0] + val[5] - val[3] + val[8] - val[6]) / 3;
    if (x <= lo_x + EDGE_EFFECT) {
      if (dx < 0)
          dx = 0;
    } else if (x >= hi_x - EDGE_EFFECT) {
      if (dx > 0)
          dx = 0;
    }
    if (y <= lo_y + EDGE_EFFECT) {
      if (dy < 0)
          dy = 0;
    } else if (y >= hi_y - EDGE_EFFECT) {
      if (dy > 0)
          dy = 0;
    }
    dy = - dy;
    delta_x = dx;
    delta_y = dy;
    level = val[4];
    octant = cartesian_to_polar(delta_x, delta_y);
}

static init_hash_table()
{
int i;

    for (i = 0; i < HASH_SIZE; lochash[i++] = NULL);
}
```

```
static int hcalls, hsteps, hmax, hpoints, hrepeats;

static loc lookuploc(x, y)
int x, y;
{
int ix, count;
loc p, *ptr;

    ++ hcalls;
    ix = (x + y) + (x << 3) - (x >> 5) + (x << 6) - (x >> 2);
    ix &= HASH_SIZE - 1;
    ptr = lochash + ix;
    count = 0;
    while (*ptr != NULL) {
      p = *ptr;
      if (p->x == x && p->y == y)
          return(p);
      ptr = &p->next;
      ++ count;
    }
    if (count > hmax)
      hmax = count;
    hsteps += count;
    ++ hpoints;
    p = new(locrec);
    *ptr = p;
    p->next = NULL;
    p->color = 0;
    p->x = x;
    p->y = y;
    p->feature_id = -1;
    p->left = empty_connections;
    p->right = empty_connections;
    p->cleft = p->cright = 0;
    p->reached = 0;
    return(p);
}

static loc stack[5000];
static int sp;

static loc outline[5000];
static int stepdir[5000];
static int step_x[8] = {1, 0, -1, -1, -1, 0, 1, 1};
static int step_y[8] = {1, 1, 1, 0, -1, -1, -1, 0};
static int absangle[8] = {0, 1, 2, 3, 4, 3, 2, 1};

#define B 4
int pgf = 0;

static loc follow_gradient(
      graphics_data *data, int x, int y, Boolean to_left, int cut)
{
int origdir, recentdir, angle;
int oc, i, j, k, newx, newy, curx, cury, max, maxoc, delta, steepness;
int lastoctant, dir, x_change, y_change, rightangle, prevmax; loc p, q,
r; short *ptr; Boolean looped; splotch s; int levels[4], level_sum; int
```

```
delx[B + 1], dely[B + 1]; connections c; char *connec; Boolean foo;
Boolean boundary;

    if (to_left != 0 && to_left != 1) {
      printf("Screwup\n");
      exit(1);
    }
    ++ feature_id;
    PIF "%d: Tracing %c from %d,%d\n",
                feature_id, "RL"[to_left], x, y ENDPIFF
    curx = x;
    cury = y;
    looped = False;
    k = 0;
    level_sum = 0;
    for (i = 0; i <= B; ++i)
      delx[i] = dely[i] = 0;
    for (i = 0; i < 4; ++i)
      levels[i] = 0;

    p = lookuploc(curx, cury);
    outline[0] = p;
    rightangle = to_left ? 2 : -2;
    c = to_left ? &p->left : &p->right;
    lastoctant = -1;
    boundary = False;
    prevmax = 50;

    for (j = 0; True; ++j) {

      ++ pgf;
      find_gradient(data, curx, cury);
      steepness = mag / 10;

      level_sum += level - levels[j & 3];
      levels[j & 3] = level;

      delx[B] += delta_x - delx[j & 3];
      dely[B] += delta_y - dely[j & 3];
      delx[j & 3] = delta_x;
      dely[j & 3] = delta_y;
      recentdir = cartesian_to_polar(delx[B], dely[B]);
      if (j >= 4)
          steepness = mag / 40;

      origdir = octant;
      if (c->which_dir < 0) {
          max = -1;
          maxoc = origdir;
          dir = (recentdir + 2) & 7;
          for (i = 0; i < 4; ++i) {
            /*
            if (i == 3 && j > 2)
                break;
            */
            if (to_left)
                oc = (origdir + i) & 7;
```

```
        else
            oc = (origdir - i - 1) & 7;
        newx = curx + step_x[oc];
        newy = cury - step_y[oc];
        if (j > 0) {
            q = outline[j - 1];
            if (newx == q->x && newy == q->y)
              continue;
        }
        find_gradient(data, newx, newy);
        if (j >= 3) {
            angle = absangle[(octant - recentdir) & 7];
            if (angle > 1) {
              if (angle > 2)
                  continue;
              mag >>= 1;
            }
            delta = (level_sum >> 2) - level;
            if (delta > 0)
              mag -= delta; // + (delta >> 1);
        }
        if (mag > max) {
            max = mag;
            maxoc = oc;
        }
    }
} else {
    maxoc = c->which_dir;
}
stepdir[j] = maxoc;
x_change = step_x[maxoc];
y_change = - step_y[maxoc];
curx = curx + x_change;
cury = cury + y_change;
lastoctant = maxoc;
p = lookuploc(curx, cury);
outline[j + 1] = p;
c = to_left ? &p->left : &p->right;
if (j < 15 || j % 10 == 0) {
    if (cfp) fprintf(cfp, "%s%s %d (%d,%d)",
            (k % 4 == 3) ? "\n\t" : " ",
            dirname[(origdir + rightangle) & 7],
            steepness, curx, cury);
    ++ k;
}
if (p->feature_id == -1 - feature_id) {
    if (! looped) {
      PIF "Detected a loop at %d,%d\n", curx, cury ENDPIF
      looped = True;
    }
} else if (p->feature_id < 0) {
    if (j >= cut)
      p->feature_id = -1 - feature_id;
} else {
    if (c->which_dir >= 0) {
    // if (p->reached & (1 << to_left)) {
      if (cfp) fprintf(cfp, "\nEntered previously taken path\n");
```

```
            if (j < 3)
                return(NULL);
            break;
          }
          ++ hrepeats;
        }

        if (looped) {
          for (i = 1; i <= j; ++i) {
            q = outline[j + 1 - i];
            if (q == p)
                break;
          }
          if (cfp) fprintf(cfp, "\nEdge forms a loop of %d points\n",
i);
          if (i < 30 && j < 50)
            return(NULL);
          break;
        }
        if (max < 40 /* && prevmax < 40 */) {
          PIF "\nEdge faded out (grad = %d) at %d,%d\n",
                                max, curx, cury ENDPIF
          if (j < 50)
            return(NULL);
          break;
        }
        if (x_change <= 0 && curx <= lo_x + 2 ||
                x_change >= 0 && curx >= hi_x - 2 ||
                y_change <= 0 && cury <= lo_y + 2 ||
                y_change >= 0 && cury >= hi_y - 2) {
          if (j > 0) {
            if (cfp) fprintf(cfp, "\nEdge reached boundary\n");
            boundary = True;
            if (j < 2)
                return(NULL);
            break;
          }
          prevmax = max;
        }
        if (j > 14) {
          q = outline[j - 13];
          if ((unsigned) (p->x - q->x + 4) <= 8 &&
                  (unsigned) (p->y - q->y + 4) <= 8) {
            PIF "\nEdge not making progress\n" ENDPIF
            if (j < 50)
                return(NULL);
            break;
          }
        }
        if (j >= 4000) {
          if (cfp) fprintf(cfp, "\nJust keeps going and going...\n");
          break;
        }
      }
      ++ j;
      if (j < cut && ! boundary)
        return;
```

```
    if (cfp) fprintf(cfp, "Adding splotch %d\n", feature_id),
fflush(cfp);
    s = new_splotch(&data->features);
    s->dir = to_left;
    s->edge = newarray(loc, j + 2);
    if (j < 7)
      i = 0;
    else if (j < cut)
      i = j / 2;
    else
      i = cut;
    for (k = 0; i <= j; ++k, ++i) {
      p = outline[i];
      s->edge[k] = p;
      c = to_left ? &p->left : &p->right;
      if (i != j) {
          connec = to_left ? &p->cleft : &p->cright;
          *connec |= 1 << stepdir[i];
          if (c->which_dir >= 0 && c->which_dir != stepdir[i]) {
            if(cfp)fprintf(cfp,"Inconsistency at %d,%d: old=%d,
new=%d\n",
                  p->x, p->y, c->which_dir, stepdir[i]);
          }
          c->which_dir = stepdir[i];
      }
      if (k != 0) {
          connec = to_left ? &p->cright : &p->cleft;
          *connec |= 1 << (stepdir[i - 1] ^ 4);
      }
      if (i > 4 || j < 7) {
          p->reached |= 1 << to_left;
      }
      if (p->feature_id < 0)
          p->feature_id = feature_id;
    }
    s->edge[k] = NULL;
    PIF "Stacking %d,%d\n", outline[j]->x, outline[j]->y ENDPIF
    stack[++sp] = outline[j];
    for (i = 40; i < j; i += 20) {
      stack[++sp] = outline[i];
      PIF "Stacking %d,%d\n", outline[i]->x, outline[i]->y ENDPIF
    }
    PIF "\n" ENDPIFF
    return(outline[j]);
}

static int bottom, top;

static int find_range(char c, int go_left)
{
int i, count;

    if (c == 0) {
      bottom = top = -1;
      return(-1);
    }
    count = 0;
```

```
    for (i = 0; i < 16; ++i) {
      if ((c >> (i & 7)) & 1) {
          if (count >= 4)
            break;
          count = 0;
      } else {
          ++ count;
      }
    }
    if (i == 16) {
      bottom = top = -2;
      return(-2);
    }
    bottom = i & 7;
    top = bottom;
    for (i = bottom + 1; i < bottom + 4; ++i) {
      if ((c >> (i & 7)) & 1)
          top = i & 7;
    }
    return(go_left ? bottom : top);
}

static int break_count;

static sever(loc p, int goleft, int dir, int depth)
{
int x, y, k;
loc q;
char *ptr, *qptr;
Boolean foo;

    foo = cfp != NULL && p->x == 485 && p->y == 231;
    ptr = goleft ? &p->cleft : &p->cright;
    if ((*ptr & (1 << dir)) == 0)
      return;
    if ((unsigned) (p->x - 493 + 3) <= 6 && (unsigned) (p->y - 230 + 3)
<= 6)
      if (cfp) fprintf(cfp, "Severing %d,%d: %c dir=%d, #=%d,
depth=%d\n",
                  p->x, p->y, "RL"[goleft], dir, break_count, depth);
    *ptr &= ~ (1 << dir);
    x = p->x + step_x[dir];
    y = p->y - step_y[dir];
    q = lookuploc(x, y);
    if (foo) fprintf(cfp, "Cutting link to %d,%d (%x<->%x,
reached=%d)\n",
            q->x, q->y, q->cleft & 255, q->cright & 255, q->reached);
    qptr = goleft ? &q->cright : &q->cleft;
    *qptr &= ~ (1 << (dir ^ 4));
    if (*qptr == 0 && q->reached != 3) {
      for (k = 0; k < 8; ++k) {
          sever(q, goleft, k, depth + 1);
      }
    }
}

#define LOOKAHEAD 10
```

```
static int pick_best_path(loc p, int goleft, int preferred, int depth)
{ int step, i, j, k, keeper, ix, x, y, going, choice, maxstep; int
dirs[4]; int prevcolor[4 * LOOKAHEAD]; loc path[4 * LOOKAHEAD]; int
route[4 * LOOKAHEAD]; int savecolor; loc q, r; char qcon, *ptr; Boolean
foo;

    if (depth > 10)
      return(-1);
    foo = NEAR(p->x, 111, 3) && NEAR(p->y, 321, 3);
    savecolor = curcolor;
    ptr = goleft ? &p->cleft : &p->cright;
    k = 0;
    if (goleft) {
      for (i = 0; i < 4; ++i) {
          j = (preferred + i) & 7;
          if (*ptr & (1 << j))
            dirs[k++] = j;
      }
    } else {
      for (i = 0; i < 4; ++i) {
          j = (preferred - i) & 7;
          if (*ptr & (1 << j))
            dirs[k++] = j;
      }
    }
    going = k;
    PIF "pick_best_path %d,%d (%c) dirs=%d, depth=%d\n",
            p->x, p->y, "RL"[goleft], going, depth ENDPIF
    for (i = 0; i < k; ++i) {
      ix = i;
      x = p->x + step_x[dirs[i]];
      y = p->y - step_y[dirs[i]];
      q = lookuploc(x, y);
      prevcolor[ix] = q->color;
      route[ix] = dirs[i];
      q->color = savecolor + ix;
      curcolor = q->color + 1;
      path[ix] = q;
    }
    for (step = 0; step < LOOKAHEAD - 1 && going > 1; ++step) {
      for (i = 0; i < k; ++i) {
          ix = i + (step + 1) * 4;
          path[ix] = NULL;
          q = path[ix - 4];
          if (q == NULL)
            continue;
          qcon = goleft ? q->cleft : q->cright;
          keeper = find_range(qcon, goleft);
          if (keeper < 0) {
            -- going;
            continue;
          }
          if (top != bottom) {
            if (q == p)
                return(-1);
            keeper = pick_best_path(q, goleft, keeper, depth + 1);
```

```
        if (keeper == -1)
            return(-1);
      }
      x = q->x + step_x[keeper];
      y = q->y - step_y[keeper];
      r = lookuploc(x, y);
      if (r->color >= savecolor) {
        -- going;
        continue;
      }
      prevcolor[ix] = r->color;
      route[ix] = keeper;
      r->color = savecolor + ix;
      curcolor = r->color + 1;
      path[ix] = r;
    }
  }
  if (going == 0)
    -- step;
  maxstep = step;
  for (choice = 0; choice < k; ++choice) {
    ix = choice + step * 4;
    if (path[ix] != NULL)
        break;
  }
  if (choice >= k)
    return(-1);
  *ptr = 1 << dirs[choice];
  curcolor = savecolor;
  for (i = 0; i < k; ++i) {
    for (step = 0; step <= maxstep; ++step) {
        ix = i + step * 4;
        q = path[ix];
        if (q != NULL)
          q->color = prevcolor[ix];
    }
  }
  for (i = 0; i < k; ++i) {
    if (i == choice)
        continue;
    for (step = 0; step <= maxstep; ++step) {
        ix = i + step * 4;
        q = path[ix];
        if (q == NULL)
          break;
        keeper = route[ix];
        ptr = goleft ? &q->cright : &q->cleft;
        *ptr &= ~ (1 << (keeper ^ 4));
        qcon = *ptr;
        // *ptr = 0;
        if (step != 0) {
          ix = i + (step - 1) * 4;
          q = path[ix];
          ptr = goleft ? &q->cleft : &q->cright;
          *ptr &= ~ (1 << keeper);
          // q->cleft = q->cright = 0;
        }
```

```
            if (qcon != 0)
              break;
          }
        }
        return(dirs[choice]);
}

static int callno;

static break_links(splotch s)
{
int i, goleft, len, keeper, j, lim, x, y;
loc p, q;
char *ptr, *qptr;
connections c, r;
Boolean foo;

        for (len = 0; s->edge[len] != NULL; ++len);
        for (i = 0; i < len; ++i) {
          p = s->edge[i];
          foo = NEAR(p->x, 171,3) && NEAR(p->y, 182, 3);
          if (foo) PIF "%d,%d connected L=%x, R=%x\n", p->x, p->y,
                            p->cleft & 255, p->cright & 255 ENDPIF
          for (goleft = 0, ptr = &p->cright; goleft <= 1;
                                  ++goleft, ptr = &p->cleft) {
              keeper = find_range(*ptr, goleft);
              if (keeper < 0)
                continue;
              if (top == bottom)
                continue;
              ++ callno;
              ++ curcolor;
              pick_best_path(p, goleft, keeper, 0);
          }
        }
}

static void detect_edges(int **suspects, graphics_data *data)
{
int i, j, *ptr, x, y, dir;
loc p;

        for (j = 0; suspects[j] != NULL; ++j) {
          ptr = suspects[j];
          for (i = 0; ptr[i] >= 0; i += 2) {
              x = ptr[i];
              y = ptr[i + 1];
              for (dir = 0; dir <= 1; ++dir) {
                p = follow_gradient(data, x, y, dir, 25);
                while (sp > 0) {
                    p = stack[sp--];
                    if (p->reached != 3)
                      (void) follow_gradient(data, p->x, p->y,
                                        2 - p->reached, 3);
                }
              }
          }
```

```
      PIF "\n" ENDPIFF
    }
}

static loc pointbuffer[10000];
static int log_2[256];

static consolidate(data)
graphics_data *data;
{
splotch s, n, lis;
int i, inc, ix, con, dir, x, y, hi, lo;
loc p, q;

    for (i = 1; i < 256; ++i)
      log_2[i] = -2;
    log_2[0] = -1;
    for (i = 0; i < 8; ++i)
      log_2[1 << i] = i;
    loop_over_list(s, data->features)
      for (i = 0; s->edge[i] != NULL; ++i)
          s->edge[i]->color = 0;
    curcolor = 1;
    lis = NULL;
    loop_over_list(s, data->features) {
      for (i = 0; s->edge[i] != NULL; ++i) {
          p = s->edge[i];
          if (p->color >= curcolor)
            continue;
          if (p->cleft == 0 && p->cright == 0)
            continue;
          ++ feature_id;
          PIF "Starting feature %d: %d,%d\n", feature_id, p->x, p->y
ENDPIF
          for (inc = -1; inc <= 1; inc += 2) {
            ix = 5000;
            pointbuffer[ix] = p;
            q = p;
            while (True) {
                q->color = curcolor + feature_id;
                q->feature_id = feature_id;
                con = inc == -1 ? q->cleft : q->cright;
                dir = log_2[con & 255];
                /*
                PIF "    %d,%d: dir = (%x) %d\n",
                             q->x, q->y, con & 255, dir ENDPIF
                */
                if (dir < 0) {
                  if (dir < -1)
                      PIF "Ambiguous point %d,%d: %c->%x\n", q->x, q-
>y,
                                 "RL"[inc == -1], con & 255 ENDPIF
                  break;
                }
                x = q->x + step_x[dir];
                y = q->y - step_y[dir];
                q = lookuploc(x, y);
```

```
                if (q->color >= curcolor) {
                  PIF "    Bailing at %d,%d due to color=%d\n",
                        x, y, q->color ENDPIF
                  break;
                }
                ix += inc;
                pointbuffer[ix] = q;
              }
              *(inc == -1 ? &lo : &hi) = ix;
            }
            n = new_splotch(&lis);
            n->edge = newarray(loc, hi - lo + 2);
            for (ix = lo; ix <= hi; ++ix)
              n->edge[ix - lo] = pointbuffer[ix];
            n->edge[ix - lo] = NULL;
            q = pointbuffer[lo];
            p = pointbuffer[hi];
            PIF "    Feature from %d(%d,%d) to %d(%d,%d)\n",
              lo, q->x, q->y, hi, p->x, p->y ENDPIFF
        }
    }
    data->features = lis;
}

static find_folds(data)
graphics_data *data;
{
splotch s, r, orig;
int i, len, dir1, dir2, dir3, angle, th1, th2, restart;
loc p, q, t, u;

    loop_over_list(orig, data->features) {
      s = orig;
      for (len = 0; s->edge[len] != NULL; ++len);
      if (len < edge_width * 3)
          continue;
      for (i = edge_width * 3; i < len; i += edge_width) {
          p = s->edge[i - edge_width * 3];
          q = s->edge[i - edge_width * 2];
          t = s->edge[i - edge_width];
          u = s->edge[i];
          dir1 = cartesian_to_polar(q->x - p->x, p->y - q->y);
          dir3 = cartesian_to_polar(u->x - t->x, t->y - u->y);
          angle = (dir3 - dir1) & 7;
          if (absangle[angle] >= 3) {
            dir2 = cartesian_to_polar(t->x - q->x, q->y - t->y);
            th1 = (dir2 - dir1) & 7;
            th2 = (dir3 - dir2) & 7;
            if (th1 >= 5 && th2 >= 5)
                continue;     // Backward fold -- no sense breaking
those

            PIF "Found a fold at %d,%d - %d,%d (dir %d -> %d -> %d)\n",
                  q->x, q->y,
                  t->x, t->y,
                  dir1, dir2, dir3 ENDPIFF
```

```
            restart = i - edge_width;

            // Terminate old splotch early
            s->edge[restart - edge_width + 1] = NULL;

            // Should hunt for the best spot to break instead of just
            // snipping middle segment, but implement it later.

            // Start a new splotch after the break
            ++ feature_id;
            r = new_splotch(&data->features);
            r->edge = &s->edge[restart];

            // Continue, looking for more folds in the new splotch
            s = r;
            len -= restart;
            i = edge_width * 2;
          }
      }
    }
}


typedef struct {
    splotch which;
    char dir;
    char len;
    loc p;
} passthrurec, *passthru;

typedef struct {
    passthrurec pt[2];
    short x, y;
} phboxrec, *phbox;

static put_in_ph(splotch s, phbox b, loc enters, loc leaves, int cnum)
{ int dx, dy; int dir, i, angle, minangle, ix; passthru pt, ot;

    dx = leaves->x - enters->x;
    dy = leaves->y - enters->y;
    dir = cartesian_to_polar(dx, - dy);
    i = dir / 2;
    if (cnum == ((i - 1) & 3))
      return;
    if (cnum == ((i - 2) & 3))
      return;
    minangle = 5;
    for (i = 0; i < 2; ++i) {
      pt = &b->pt[i];
      if (pt->which == s)
          angle = 0;
      else if (pt->len == 0)
          angle = 2;
      else
          angle = absangle[(dir - pt->dir) & 7];
      if (minangle > angle) {
          minangle = angle;
```

```
            ix = i;
          }
        }
        pt = &b->pt[ix];
        if (mag > pt->len) {
          PIF "Pigeoning %d,%d to %d,%d: s%d (%d @ %d [%d,%d])\n",
                enters->x, enters->y, leaves->x, leaves->y,
                s->id, mag, dir, dx, -dy ENDPIF
          pt->len = mag;
          pt->dir = dir;
          pt->which = s;
          pt->p = enters;
        }
    }

    static splotch chase_mesas(splotch s)
    {
    splotch p;

        for (p = s; p != NULL; p = p->mesa)
          if (p->mesa == NULL || p->mesa == p)
              break;
        return(p);
    }

    static group(s1, s2)
    splotch s1, s2;
    {
    splotch r, p;

        r = chase_mesas(s1);
        p = chase_mesas(s2);
        PIF "Grouping %d(%d) with %d(%d)\n", s1->id, r->id, s2->id, p->id
    ENDPIF
        if (p->id < r->id)
          r->mesa = p;
        else
          p->mesa = r;
    }

    static record_pairing(phbox box)
    {
    pairing p;
    splotch s1, s2;

        s1 = box->pt[0].which;
        s2 = box->pt[1].which;
        if (s1->id > s2->id) {
          s1 = box->pt[1].which;
          s2 = box->pt[0].which;
        }
        loop_over_list(p, s1->partners) {
          if (p->s1 == s1 && p->s2 == s2)
              break;
        }
        if (p == NULL) {
          p = new(pairingrec);
```

```
      p->next = s1->partners;
      s1->partners = p;
      p->s1 = s1;
      p->s2 = s2;
      p->count = 0;
    }
    ++ p->count;
    if (p->count == 2)
      group(s1, s2);
}

static pairing last_partner(pairing p)
{
    for ( ; p->next != NULL; p = p->next);
    return(p);
}

static bipartition(splotch s)
{
pairing p, q;
int max, p1, p2, newp1, newp2;
Boolean useful;

    q = NULL;
    max = 0;
    loop_over_list(p, s->partners)
      if (max < p->count) {
          max = p->count;
          q = p;
      }
    if (max < 2)
      return;
    q->s1->polarity = max;
    q->s2->polarity = - max;
    useful = True;
    while (useful) {
      useful = False;
      loop_over_list(q, s->partners) {
          newp1 = newp2 = 0;
          p1 = q->s1->polarity;
          p2 = q->s2->polarity;
          if (p1 == 0) {
            newp1 = - p2;
          } else if (p2 == 0) {
            newp2 = - p1;
          } else if ((p1 > 0) == (p2 > 0)) {
            if ((p1 > 0) ? p1 > p2 : p1 < p2)
                newp2 = - p1;
            else
                newp1 = - p2;
          } else {
            if ((p1 > 0) ? p1 > - p2 : p1 < - p2)
                newp2 = - p1;
            else
                newp1 = - p2;
          }
          if (newp1 > q->count) newp1 = q->count;
```

```
            if (newp1 < - q->count) newp1 = - q->count;
            if (newp2 > q->count) newp2 = q->count;
            if (newp2 < - q->count) newp2 = - q->count;
            if (abs(p1) < abs(newp1)) {
              q->s1->polarity = newp1;
              useful = True;
            }
            if (abs(p2) < abs(newp2)) {
              q->s2->polarity = newp2;
              useful = True;
            }
        }
    }
}

static pigeon_hole(data)
graphics_data *data;
{
splotch s;
int nx, ny, i, x, y, ix, iy, j, k, id, cnum;
phbox boxes, box, corner[5];
int stepx[5], stepy[5];
loc p, entry[5];
passthru pt;

    PIF "Pigeonholing\n" ENDPIFF
    nx = (hi_x - lo_x - 1) / box_size + 1;
    ny = (hi_y - lo_y - 1) / box_size + 1;
    boxes = newarray(phboxrec, nx * ny);
    for (i = nx * ny; --i >= 0; ) {
      box = &boxes[i];
      box->pt[0].len = box->pt[1].len = 0;
      box->x = i % nx * box_size + lo_x;
      box->y = i / nx * box_size + lo_y;
    }
    for (i = 0; i < 4; ++i) {
      stepx[i] = step_x[i * 2 + 1] * edge_width;
      stepy[i] = step_y[i * 2 + 1] * edge_width;
    }
    stepx[4] = stepy[4] = 0;
    loop_over_list(s, data->features) {
      id = s->id;
      for (cnum = 0; cnum < 5; corner[cnum++] = NULL);
      for (i = 0; s->edge[i] != NULL; ++i) {
          p = s->edge[i];
          for (cnum = 0; cnum < 5; cnum++) {
            x = (p->x - lo_x + stepx[cnum]) / box_size;
            y = (p->y - lo_y - stepy[cnum]) / box_size;
            if (x >= nx || y >= ny)
                box = NULL;
            else
                box = &boxes[y * nx + x];
            if (box != corner[cnum]) {
                if (corner[cnum] != NULL) {
                  put_in_ph(s, corner[cnum], entry[cnum], p, cnum);
                }
                corner[cnum] = box;
```

```
                entry[cnum] = p;
              }
            }
        }
        for (cnum = 0; cnum < 5; cnum++) {
            if (corner[cnum] != NULL) {
              put_in_ph(s, corner[cnum], entry[cnum], p, cnum);
            }
        }
      }
      for (y = 0; y < ny; ++y) {
        for (x = 0; x < nx; ++x) {
            box = &boxes[y * nx + x];
            if (box->pt[0].len == 0)
              continue;
            PIF "%d,%d - %d,%d:",
              x * box_size + lo_x, y * box_size + lo_y,
              (x + 1) * box_size + lo_x, (y + 1) * box_size + lo_y
              ENDPIF
            for (i = 0; i < 2; ++i) {
              pt = &box->pt[i];
              if (pt->len == 0)
                  break;
              PIF " (s%d dir=%d, len=%d)",
                          pt->which->id, pt->dir, pt->len ENDPIF
            }
            PIF "\n" ENDPIF
            if (i == 2)
              record_pairing(box);
        }
      }
      loop_over_list(s, data->features)
        s->mesa = chase_mesas(s);
      loop_over_list(s, data->features)
        if (s->mesa != s)
            if (s->partners != NULL) {
              last_partner(s->partners)->next = s->mesa->partners;
              s->mesa->partners = s->partners;
              s->partners = NULL;
            }
      loop_over_list(s, data->features)
        if (s->mesa == s)
            bipartition(s);
      loop_over_list(s, data->features) {
        PIF "Splotch %d is in mesa %d (pol=%d)\n",
                    s->id, s->mesa->id, s->polarity ENDPIF
      }
}

static orient_mesa(splotch *pile)
{
int i, j, k, pos, plus_inside, minus_inside, cutoff;
int *min, *max, *which;
splotch s;
int plus_bbox[4], minus_bbox[4], diff[4];
loc p;
```

```
    for (i = 0, which = plus_bbox; i < 2; ++i, which = minus_bbox) {
      min = which;
      max = which + 2;
      min[0] = min[1] = 1000000;
      max[0] = max[1] = -1;
    }
    for (i = 0; pile[i] != NULL; ++i) {
      s = pile[i];
      which = s->polarity > 0 ? plus_bbox : minus_bbox;
      min = which;
      max = which + 2;
      for (j = 0; s->edge[j] != NULL; ++j) {
          p = s->edge[j];
          for (k = 0, pos = p->x; k < 2; ++k, pos = p->y) {
            if (min[k] > pos)
                min[k] = pos;
            if (max[k] < pos)
                max[k] = pos;
          }
      }
    }
    for (i = 0; i < 4; ++i)
      diff[i] = plus_bbox[i] - minus_bbox[i];
    for (i = 2; i < 4; ++i)
      diff[i] = - diff[i];
    cutoff = 2;
    plus_inside = minus_inside = 0;
    for (i = 0; i < 4; ++i)
      if (diff[i] < - cutoff)
          ++ minus_inside;
      else if (diff[i] > cutoff)
          ++ plus_inside;
    which = plus_inside < minus_inside ? plus_bbox : minus_bbox;
    s = pile[0]->mesa;
    PIF "Bounding box of mesa %d is %d,%d to %d,%d\n", s->id,
            which[0], which[1], which[2], which[3] ENDPIF
    PIF "    Red inside: %d, Blue inside: %d\n",
                  plus_inside, minus_inside ENDPIFF
    if (minus_inside > plus_inside) {
      PIF "    Reversing\n" ENDPIFF
      for (i = 0; pile[i] != NULL; ++i) {
          s = pile[i];
          s->polarity = - s->polarity;
      }
    }
}

static decide_inside_outside(data)
graphics_data *data;
{
splotch s, t;
int count;
splotch pile[100];

    loop_over_list(s, data->features)
      if (s->mesa == s) {
        count = 0;
```

```
            loop_over_list(t, data->features)
              if (t->mesa == s)
                  if (t->polarity <= -2 || t->polarity >= 2)
                    pile[count++] = t;
            pile[count] = NULL;
            if (count >= 2)
              orient_mesa(pile);
        }
}

examine_cross_sections(graphics_data *data)
{
windo sem;
int xsize, ysize, count, i, j;
double f, separation;
splotch s;
int *suspects[50];


    data->features = NULL;
    if ((int) cfp == -1)
      cfp = fopen("cross_section", "w");

    theDisp=XtDisplay(data->sem->thiscanvas);
    data->features = NULL;
    init_hash_table();
    identify_border_areas(data);
    sem = data->sem;
    xsize = hi_x - lo_x;
    ysize = hi_y - lo_y;

    separation = 1.0 / 6;
    for (i = 0; i <= HIST_SIZE; histogram[i++] = 0);
    count = 0;
    for (f = separation * 0.5; f < 0.99; f += separation)
      suspects[count++] = cross_section(data, lo_y + (int) (ysize * f),
1);
    for (f = separation * 0.5; f < 0.99; f += separation)
      suspects[count++] = cross_section(data, lo_x + (int) (xsize * f),
0);
    suspects[count] = NULL;
    analyze_pair_statistics();
    detect_edges(suspects, data);
    loop_over_list(s, data->features)
      break_links(s);
    consolidate(data);
    find_folds(data);
    pigeon_hole(data);
    decide_inside_outside(data);
    PIF "\n" ENDPIFF
    /*
    printf("hcalls = %d, hpoints = %d, hsteps = %d, hmax = %d\n",
            hcalls, hpoints, hsteps, hmax);
    printf("hrepeats = %d\n", hrepeats);
    */
}
```

---------- Subclaim 1-e -- excerpt from callbacks.c -- May 25, 1999 --
----

This file contains a procedure for drawing the contours of an alim image (an aerial image or latent image) onto a digitized wafer pattern image. When both the alim contours and the detected pattern edges are drawn onto the same pattern image, they are overlaid on each other.

```
/*
%   Various Callback Routines and Drawing Functions for SmartSEM
%
%                          Dec. 1995
*/

#include <stdio.h>
#include <math.h>
#include "Screens.h"
#include "plot.h"

extern int           DrawFillMaskFlag, DrawGrid;
extern MaskRegion    mrd;
extern int           npregions;
extern int           DrawImageInMaskFlag;
extern Pixel red_pixel;
extern int
color_map_type,color_mask,color_label,color_SEM_contour;
extern int           width_label,width_SEM_contour,width_mask;
extern double contour_value;
extern double contour_list[];

extern windo newwindo();

Widget          trackerP,     /* widgets for tracking cursor      */
                trackerW,     /* pixel, world x,y and       */
                valueDisplay, /* function min and max values      */
                trackerD;
                              /* function value at point*/

static graphics_data  *contour_data;
static Widget contour_widget;

int ixoffset, iyoffset, y_top;
double x_factor, y_factor;


/********************** Contour Drawing Functions
**********************/

#include <sys/time.h>
hrtime_t start, end;

static int pixcount;
static int matrix_size = 0;
static float *z_matrix;
static Display *contour_dpy;
static float xavg, yavg, zavg;
```

```
static Boolean doSIM;
static float *x, *y;


#define FAREF(i,j) ((float) z_matrix[(j)*nx+(i)])

world_to_pixel_conversion(graphics_data *data, Boolean sem)
{
windo w;

    if (sem) {
      w = data->sem;
      y_top = w->yres;
    } else {
      w = data->aim;
      y_top = data->height;
    }
    x_factor = 1.0 / w->pixel.x;
    y_factor = 1.0 / w->pixel.y;
    ixoffset =   (int) (w->ll.x * x_factor + 0.5);
    iyoffset = - (int) (w->ll.y * y_factor + 0.5);
}

      /* Draw a line between two points, given pixel coordinates. */
static void contour_seg(int ix2, int iy2, int ix, int iy) {
    XDrawLine(contour_dpy, XtWindow(contour_widget),
            contour_data->gcSEMContour,
            ix2,iy2,ix,iy);
}

      /* Draw a line between two points, given world coordinates. */
static connect_points(x1, y1, x2, y2) double x1, y1, x2, y2; { int ix,
iy, ix2, iy2;


    ix = (int) (x1 * x_factor + 0.5) - ixoffset;
    iy = y_top - (int) (y1 * y_factor + 0.5) - iyoffset;
    ix2 = (int) (x2 * x_factor + 0.5) - ixoffset;
    iy2 = y_top - (int) (y2 * y_factor + 0.5) - iyoffset;

    contour_seg(ix2, iy2, ix, iy);
}

      /* Draw the line at a given z-value across a triangle. */ static
int plotContourSegment(float level, float x1,float y1,float z1,
                                    float x2, float y2,float z2)
{
float x3, y3, z3;
graphics_data  *data;
float x, y, factor, temp;
int   ix,iy,ix2,iy2;
float xt,yt;

    data = contour_data;
    x3 = xavg;
    y3 = yavg;
    z3 = zavg;
```

```
if(z1 >= z2) {
  temp = x1; x1 = x2; x2 = temp;
  temp = y1; y1 = y2; y2 = temp;
  temp = z1; z1 = z2; z2 = temp;
}
if(z2 > z3) {
  if(z3 < z1) {
      temp = x1; x1 = x3; x3 = temp;
      temp = y1; y1 = y3; y3 = temp;
      temp = z1; z1 = z3; z3 = temp;
  }
  temp = x2; x2 = x3; x3 = temp;
  temp = y2; y2 = y3; y3 = temp;
  temp = z2; z2 = z3; z3 = temp;
}
/* z1 <= z2 <= z3 */
if(level < z1 || level > z3) {
  return -1;
}
if((level == z1) && (z1 == z2) && (z2 != z3)) {
  /* draw a line from x1, y1 to x2, y2 */
  connect_points(x1, y1, x2, y2);
  return 1;
}
if((level == z3) && (z3 == z2) && (z1 != z2)) {
  /* draw a line from x2, y2 to x3, y3 */
  connect_points(x3, y3, x2, y2);
  return 1;
}

if(level == z2) {
  /* we know that z2 is definitely between z1 and z3
     draw a line from P2 to point on segment between P1 and P3
   */
  if(z3 == z1)
      return 0;
  factor = (level - z1)/(z3-z1);
  x = x1 + (x3-x1) * factor;
  y = y1 + (y3-y1) * factor;
  connect_points(x, y, x2, y2);
  return 1;
}
if(level < z2) {
  if(z2 == z1 || z3 == z1)
      return 0;
  /* P2 and P3 are above, P1 below */
  factor = (level - z1) / (z2 - z1);
  xt = x1 + (x2 - x1) * factor;
  yt = y1 + (y2 - y1) * factor;

  factor = (level - z1) / (z3 - z1);
  x = x1 + (x3 - x1) * factor;
  y = y1 + (y3 - y1) * factor;
  connect_points(x, y, xt, yt);
} else {
```

```
        if(z3 == z1 || z3 == z2)
            return 0;
        /* P1 and P2 are below, P3 above */
        factor = (level - z1) / (z3 - z1);
        xt = x1 + (x3 - x1) * factor;
        yt = y1 + (y3 - y1) * factor;

        factor = (level - z2) / (z3 - z2);
        x = x2 + (x3 - x2) * factor;
        y = y2 + (y3 - y2) * factor;
        connect_points(x, y, xt, yt);
    }
    return 1;
}

        /* This system draws a contour line over a display square by
           identifying which sides of the square cross the contour
           line.  If zero do, there's no line to draw.  If all four
           do, this method doesn't work, so Bob Pack's old pyramid
           method is used.  But if two sides cross the line, this
           technique identifies them by table-lookup based on which
           corners are above and below the contour level, and then
           draws a line between the two crossing points.  pgf, 4/99 */

#define NO_CONTOUR 5, 5, 5, 5
#define PYRAMID 4, 4, 4, 4

typedef struct {
    int lo1, hi1, lo2, hi2;
} edgepairrec;

static edgepairrec edgepairs[17] = {            //   Corner Numbers
                        // Corner: 3210     --------------------
        NO_CONTOUR,             // LLLL          2     3
        1, 0, 2, 0,             // LLLH
        0, 1, 3, 1,             // LLHL          0     1
        2, 0, 3, 1,             // LLHH    --------------------
        0, 2, 3, 2,             // LHLL
        3, 2, 1, 0,             // LHLH
        PYRAMID,                // LHHL
        3, 2, 3, 1,             // LHHH
        1, 3, 2, 3,             // HLLL
        PYRAMID,                // HLLH
        0, 1, 2, 3,             // HLHL
        2, 0, 2, 3,             // HLHH
        0, 2, 1, 3,             // HHLL
        1, 0, 1, 3,             // HHLH
        0, 1, 0, 2,             // HHHL
        NO_CONTOUR,             // HHHH
        -1};

        /* Find the points where a contour level crosses two edges of
           a display square, and draw a line between them. */
static connect_edges(edgepairrec *ep, int i, int j, double level) { int
ixlo, ixhi, iylo, iyhi; double xlo, xhi, ylo, yhi, zlo, zhi, frac;
double x1, y1, x2, y2;
```

```
    ixlo = i + (ep->lo1 & 1);
    iylo = j + (ep->lo1 >> 1);
    ixhi = i + (ep->hi1 & 1);
    iyhi = j + (ep->hi1 >> 1);
    xlo = x[ixlo];
    xhi = x[ixhi];
    ylo = y[iylo];
    yhi = y[iyhi];
    zlo = FAREF(ixlo, iylo);
    zhi = FAREF(ixhi, iyhi);
    frac = (level - zlo) / (zhi - zlo);
    if (frac > 1.0 || frac < 0.0)
      return;
    x1 = xlo + frac * (xhi - xlo);
    y1 = ylo + frac * (yhi - ylo);

    ixlo = i + (ep->lo2 & 1);
    iylo = j + (ep->lo2 >> 1);
    ixhi = i + (ep->hi2 & 1);
    iyhi = j + (ep->hi2 >> 1);
    xlo = x[ixlo];
    xhi = x[ixhi];
    ylo = y[iylo];
    yhi = y[iyhi];
    zlo = FAREF(ixlo, iylo);
    zhi = FAREF(ixhi, iyhi);
    frac = (level - zlo) / (zhi - zlo);
    if (frac > 1.0 || frac < 0.0)
      return;
    x2 = xlo + frac * (xhi - xlo);
    y2 = ylo + frac * (yhi - ylo);

    connect_points(x1, y1, x2, y2);
}

      /* Draw the segments for a set of contour lines that pass through
         a one-grid-wide vertical stripe of an AIM or SEM. */
static draw_contour_set(double *contourset, int i, Boolean drawgrid) {
Point_ptr data_ptr; int j, k;
float     level,zij,zipj,zijp,zipjp;
double contour2[2];
int top_above[100], bottom_above[100];
int shape;
edgepairrec *ep;
windo w;

    for (j = 0; j < ny - 1; j++)
      if (y[j+1] > contour_data->aim->ll.y)
          break;
    zijp = FAREF(i, j);
    zipjp = FAREF(i+1, j);
    for (k = 0; contourset[k] > 0.0; ++k) {
      level = contourset[k];
      top_above[k] = (zijp > level) | ((zipjp > level) << 1);
    }
    for ( ; j < ny - 1; j++) {
      if (y[j] >= contour_data->aim->ur.y)
```

```
            break;
        zij = zijp;
        zipj = zipjp;
        zijp  = FAREF(i, j+1);
        zipjp = FAREF(i+1, j+1);

        yavg = (y[j] + y[j+1])/2.0;
        zavg = (zij+zipj+zijp+zipjp)/4.0;
        for (k = 0; contourset[k] > 0.0; ++k) {
            level = contourset[k];
              /* Determine which corners of the current display square
                 are above the contour level and which are below. */
            bottom_above[k] = top_above[k];
            top_above[k] = (zijp > level) | ((zipjp > level) << 1);
            shape = bottom_above[k] | (top_above[k] << 2);
            ep = &edgepairs[shape];

            if (ep->lo1 == 5)
              continue;
            if (ep->lo1 == 4 || pixcount > 15) {
              /* Pyramid method.  Chop the display square into four
                 triangles and compute the contour line for each. */
              plotContourSegment(level,
                         x[i],  y[j],  zij,  x[i],  y[j+1],zijp);
              plotContourSegment(level,
                         x[i],  y[j],  zij,  x[i+1],y[j],  zipj);
              plotContourSegment(level,
                         x[i+1],y[j],  zipj, x[i+1],y[j+1],zipjp);
              plotContourSegment(level,
                         x[i],  y[j+1],zijp, x[i+1],y[j+1],zipjp);
            } else {
              connect_edges(ep, i, j, level);
            }
        }
        if (drawgrid) {
            if (doSIM)
              w = contour_data->aim;
            else
              w = contour_data->sem;
            connect_points(x[i+1] - w->pixel.y, y[j+1], x[i+1], y[j+1]);
        }
    }
}

/*
*  Draw Contour Lines  RC Pack Nov 1997
*/
void drawContours (Widget w, graphics_data  *data, int callFunction) {
Point_ptr data_ptr;
int       i,ix,iy;
float     xlast,ylast;
Boolean     generate1, generate2;
double contour2[2];

    if (data->image == NULL || w == NULL)
      return;
    doSIM = callFunction != 1;
```

```
if (doSIM) {
  // generate1 = data->drawContourOnSIM1;
  // generate2 = data->drawContourOnSIM2;
  generate1 = XmToggleButtonGetState(data->aim->dependents[1]);
  generate2 = XmToggleButtonGetState(data->aim->dependents[3]);
} else {
  generate1 = XmToggleButtonGetState(data->sem->dependents[1]);
  generate2 = XmToggleButtonGetState(data->sem->dependents[2]);
}
if (! generate1 && ! generate2)
  return;

switch_image(data, 5);

contour_data = data;
contour_widget = w;
contour_dpy = XtDisplay(w);
if (generate1)
  parse_contour_list(data, generate2);

if (XtIsRealized(w)) {
  // 11Apr98
  if (width_SEM_contour==-1)
      width_SEM_contour=0; // A kludge until initalized
  XSetLineAttributes(contour_dpy, data->gcSEMContour,
        (unsigned) width_SEM_contour, LineSolid,CapButt,JoinMiter);
}

x = malloc (sizeof(float) * nx);
y = malloc (sizeof(float) * ny);
if (nx * ny > matrix_size) {
  matrix_size = nx * ny;
  z_matrix = malloc (sizeof(float) * matrix_size);
}

xlast = ylast = -9999.0;
ix=iy=0;

data_ptr = data->image->aimvalues;
for (i=0; i<nx*ny ; i++){
  z_matrix[i]=data_ptr->z;
  if (data_ptr->x > xlast)
      xlast=x[ix++]=data_ptr->x;
  if (data_ptr->y > ylast)
      ylast=y[iy++]=data_ptr->y;
  data_ptr++;
}

contour2[0] = data->contour_value;
contour2[1] = -1.0;

if (doSIM)
  pixcount = data->image->del.y / contour_data->aim->pixel.y;
else
  pixcount = data->image->del.y / contour_data->sem->pixel.y;

world_to_pixel_conversion(data, ! doSIM);
```

```
    // Contour Plot
    for (i = 0; i < nx - 1; i++)
      if (x[i+1] > data->aim->ll.x)
          break;
    start = gethrtime();
    for ( ; i < nx - 1; i++) {
      if (x[i] >= data->aim->ur.x)
          break;
      xavg = (x[i] + x[i+1]) / 2.0;

      if (generate1) {
          XSetForeground(contour_dpy, data->gcSEMContour,
color_SEM_contour);
          draw_contour_set(contour_list, i, DrawGrid);
      }
      if (generate2) {
          XSetForeground(contour_dpy, data->gcSEMContour, red_pixel);
          draw_contour_set(contour2, i, DrawGrid && ! generate1);
      }
    }
    end = gethrtime();
//    printf("Display time = %.2f msec\n", (end - start) * 1.0e-6);
}
```

---------- Subclaim 1-f -- excerpt from semedge.c -- June 1, 1999 ----
----

The following addition to semedge.c measures and reports the
average
and root-mean-square distances from the contours in an alim image
to the detected edges.

```
/************************ SEM/AIM Comparison Functions
***********************/

static transform_to_orig(x1, y1, px, py, data)
double x1, y1;
int *px, *py;
graphics_data *data;
{
double x, y;
int lev;

    x = (x1 * x_factor) - (ixoffset + sem_offset);
    y = y_top - (iyoffset + sem_offset) - (y1 * y_factor);
    if (data->flipX)
      x = data->sem->xres - x;
    if (data->flipY)
      y = data->sem->yres - y;
    lev = data->curpm->zoom_level;
    x = x * ZOOM_STEPS / lev;
    y = y * ZOOM_STEPS / lev;
    *px = (int) (x + 0.5);
    *py = (int) (y + 0.5);
}
```

```
static find_nearby_snakes(float *seg, graphics_data *data)
{
double sx1, sy1, sx2, sy2;
int x1, y1, x2, y2;
int ix, iy, mx, my, x, y, i, j, count, length;
int lox, loy, hix, hiy;
int nx, ny;
int minmag1, minmag2;
phbox box;
snake s;
loc p;
int distance, squaresum, metric;

    sx1 = seg[0];
    sy1 = seg[1];
    sx2 = seg[2];
    sy2 = seg[3];

    transform_to_orig(sx1, sy1, &x1, &y1, data);
    transform_to_orig(sx2, sy2, &x2, &y2, data);

    if (x1 == x2 && y1 == y2)
      return;

    cartesian_to_polar(x1 - x2, y1 - y2);
    length = mag;
    nx = (hi_x - lo_x - 1) / box_size + 1;
    ny = (hi_y - lo_y - 1) / box_size + 1;
    mx = (x1 + x2) >> 1;
    my = (y1 + y2) >> 1;
    ix = (mx - lo_x) / box_size;
    iy = (my - lo_y) / box_size;
    lox = (ix - RANGE) * box_size + lo_x;
    loy = (iy - RANGE) * box_size + lo_y;
    hix = lox + (2 * RANGE + 1) * box_size;
    hiy = loy + (2 * RANGE + 1) * box_size;
    PIF "Segment %g,%g - %g,%g uM (Sem-edge database coords: %d,%d -
%d,%d)\n",
                          sx1, sy1, sx2, sy2,
                          x1, y1, x2, y2
                          ENDPIF
    count = 0;
    for (x = ix - RANGE; x <= ix + RANGE; ++x) {
      if (x < 0 || x >= nx)
          continue;
      for (y = iy - RANGE; y <= iy + RANGE; ++y) {
          if (y < 0 || y >= ny)
            continue;
          box = &boxes[y * nx + x];
          for (i = 0; i < 2; ++i) {
            if (box->pt[i].len == 0)
                break;
            s = box->pt[i].which;
            if (s->dir != 0 || s->polarity > -2)
                continue;
            semedge[s->dir = ++count] = s;
          }
```

```
      }
    }
    lox = (x1 < x2 ? x1 : x2) - box_size;
    hix = (x1 > x2 ? x1 : x2) + box_size;
    loy = (y1 < y2 ? y1 : y2) - box_size;
    hiy = (y1 > y2 ? y1 : y2) + box_size;
    distance = 1000;
    squaresum = 1000000;
    for (i = 1; i <= count; ++i) {
      s = semedge[i];
      s->dir = 0;
      minmag1 = minmag2 = 1000000;
      for (j = 0; s->edge[j] != NULL; ++j) {
          p = s->edge[j];
          if (p->x < lox || p->x > hix || p->y < loy || p->y > hiy)
            continue;
          cartesian_to_polar(p->x - x1, p->y - y1);
          if (mag < minmag1)
            minmag1 = mag;
          cartesian_to_polar(p->x - x2, p->y - y2);
          if (mag < minmag2)
            minmag2 = mag;
      }
      PIF "    distance to s%d: %d %d\n", s->id, minmag1, minmag2
ENDPIF
      metric = minmag2 + minmag1;
      if (distance > metric)
          distance = metric;
      metric = (minmag2 - minmag1);
      metric = metric * metric + minmag2 * minmag1 * 3;
      if (squaresum > metric)
          squaresum = metric;
    }
    if (distance > 100)
      return;
    totallength += length;
    totalarea += length * distance;
    totalsquares += length * squaresum;
}

static void measure_edge_contour_difference(graphics_data *data) {
snake s; int i, segments; float *seg; double gap, pixsize;

    if (data->features->how_close >= 0.0) // if already computed
      return;
    boxes = data->features->boxes;
    printf("\nComputing the gap between the edges and the contour
lines...\n");
    for (i = 0; contour_store[i] > -1.e6 ||
            contour_store[i + 1] > -1.0e6; i += 2) {
    }
    segments = i / 4;
    totallength = totalarea = totalsquares = 0;
    for (i = 0; i < segments; ++i) {
      seg = &contour_store[i * 4];
      find_nearby_snakes(seg, data);
    }
```

```
    pixsize = (data->sem->pixel.x + data->sem->pixel.y) / 2;
    totalarea >>= 1;
    totalsquares /= 3;
    gap = sqrt((double) totalsquares / totallength);
    gap = gap * data->curpm->zoom_level / ZOOM_STEPS;
    data->features->how_close = gap * pixsize * 1000;
    printf("    ...RMS separation = %.1f nM (%.2f pixels)\n",
            data->features->how_close, gap);

    gap = (double) totalarea / totallength;
    gap = gap * data->curpm->zoom_level / ZOOM_STEPS;
    PIF "Average separation = %.1f nM (%.2f pixels)\n",
                              gap * pixsize * 1000, gap ENDPIF
    printf("    ...Average separation = %.1f nM (%.2f pixels)\n",
            gap * pixsize * 1000, gap);
}

report_edge_contour_difference(graphics_data *data)
{
char report[80];

    measure_edge_contour_difference(data);
    sprintf(report, "RMS separation = %.1f nM", data->features-
>how_close);
    show_if_changed("C", report);
}

clear_how_close(graphics_data *data)
{
    if (data->features == NULL)
      return;
    data->features->how_close = -1.0;
}

---------- Subclaim 1-g -- excerpt from semedge.c -- Feb. 29, 2000 ---
---------

    In this and subsequent versions, measure_edge_contour_difference()
    prints Molotof's best guess at what the next iteration's value of
    the alim image contour should be.  The contour is the scaled
    reciprocal of a mask processing parameter -- the amount of light
    to expose the wafer with.

static int f_cartesian_to_polar(dx, dy)
double dx, dy;
{
static int octant_map[8] = {0, 1, 3, 2, 7, 6, 4, 5};
double adx, ady;
int ix;

    if (dx < 0)
      ix = 2, adx = - dx;
    else
      ix = 0, adx = dx;
    if (dy < 0)
      ix += 4, ady = - dy;
    else
```

```
      ady = dy;
    if (adx < ady)
      ++ ix, adx *= 0.5;
    else
      ady *= 0.5;
    fmag = adx + ady;
    return(octant_map[ix]);
}

static transform_from_orig(x1, y1, px, py, data)
double x1, y1;
double *px, *py;
graphics_data *data;
{
double x, y;
int lev;

    lev = data->curpm->zoom_level;
    x = x1 * lev / (double) ZOOM_STEPS;
    y = y1 * lev / (double) ZOOM_STEPS;
    if (data->flipX)
      x = data->sem->xres - x;
    if (data->flipY)
      y = data->sem->yres - y;
    x += (ixoffset + sem_offset);
    y -= y_top - (iyoffset + sem_offset);
    x /= x_factor;
    y /= - y_factor;
    *px = x;
    *py = y;
}

static find_nearby_snakes(float *seg, graphics_data *data)
{
double sx1, sy1, sx2, sy2;
double x1, y1, x2, y2, mx, my, length;
int ix, iy, x, y, i, j, count, dir;
int lox, loy, hix, hiy;
int nx, ny;
double minmag1, minmag2;
phbox box;
snake s;
loc p, p1, p2, cp1, cp2;
double distance, squaresum, metric, z, slope1, slope2, pull_in,
push_out; double dzdx, dzdy, grad;

    sx1 = seg[0];
    sy1 = seg[1];
    sx2 = seg[2];
    sy2 = seg[3];

    transform_to_orig(sx1, sy1, &x1, &y1, data);
    transform_to_orig(sx2, sy2, &x2, &y2, data);
    dir = f_cartesian_to_polar(x1 - x2, y1 - y2);
    length = fmag;
    if (length < 0.001) // length is in pixels.
      return;
```

```
    nx = (hi_x - lo_x - 1) / box_size + 1;
    ny = (hi_y - lo_y - 1) / box_size + 1;
    mx = (x1 + x2) * 0.5;
    my = (y1 + y2) * 0.5;
    ix = (int) ((mx - lo_x) / box_size);
    iy = (int) ((my - lo_y) / box_size);
    lox = (ix - RANGE) * box_size + lo_x;
    loy = (iy - RANGE) * box_size + lo_y;
    hix = lox + (2 * RANGE + 1) * box_size;
    hiy = loy + (2 * RANGE + 1) * box_size;
    PIF "Segment %g,%g - %g,%g uM (Sem-edge database coords: %g,%g -
%g,%g)\n",
                        sx1, sy1, sx2, sy2,
                        x1, y1, x2, y2
                        ENDPIF
    count = 0;
    for (x = ix - RANGE; x <= ix + RANGE; ++x) {
      if (x < 0 || x >= nx)
          continue;
      for (y = iy - RANGE; y <= iy + RANGE; ++y) {
          if (y < 0 || y >= ny)
            continue;
          box = &boxes[y * nx + x];
          for (i = 0; i < 2; ++i) {
            if (box->pt[i].len == 0)
                break;
            s = box->pt[i].which;
            if (s->dir != 0 || s->polarity > -2)
                continue;
            semedge[s->dir = ++count] = s;
          }
      }
    }
    lox = (int) (x1 < x2 ? x1 : x2) - box_size;
    hix = (int) (x1 > x2 ? x1 : x2) + box_size;
    loy = (int) (y1 < y2 ? y1 : y2) - box_size;
    hiy = (int) (y1 > y2 ? y1 : y2) + box_size;
    distance = 1000;
    squaresum = 1000000;
    p1 = p2 = cp1 = cp2 = NULL;
    for (i = 1; i <= count; ++i) {
      s = semedge[i];
      s->dir = 0;
      minmag1 = minmag2 = 1000000.0;
      for (j = 0; s->edge[j] != NULL; ++j) {
          p = s->edge[j];
          if (p->x < lox || p->x > hix || p->y < loy || p->y > hiy)
            continue;
          f_cartesian_to_polar(p->x - x1, p->y - y1);
          if (fmag < minmag1) {
            minmag1 = fmag;
            cp1 = p;
          }
          f_cartesian_to_polar(p->x - x2, p->y - y2);
          if (fmag < minmag2) {
            minmag2 = fmag;
            cp2 = p;
```

```
            }
        }
        PIF "    distance to s%d: %d %d\n", s->id, minmag1, minmag2
ENDPIF
        metric = minmag2 + minmag1;
        if (metric < distance)
            distance = metric;
        metric = (minmag2 - minmag1);
        metric = metric * metric + minmag2 * minmag1 * 3;
        if (metric < squaresum) {
            squaresum = metric;
            p1 = cp1;
            p2 = cp2;
        }
    }
    if (distance > 100)
      return;
    accumulate_vector(x1, y1, p1->x - x1, p1->y - y1, length, dir);
    accumulate_vector(x2, y2, p2->x - x2, p2->y - y2, length, dir);
    totallength += length;
    totalarea += length * distance;
    totalsquares += length * squaresum;

    f_cartesian_to_polar(p1->x - x1, p1->y - y1);
    minmag1 = fmag;
    f_cartesian_to_polar(p2->x - x2, p2->y - y2);
    minmag2 = fmag;

    transform_from_orig((double) p1->x, (double) p1->y, &x1, &y1,
data);
    transform_from_orig((double) p2->x, (double) p2->y, &x2, &y2,
data);
    x1 = (x1 + x2) * 0.5;
    y1 = (y1 + y2) * 0.5;
    z = aim_intensity(data->image, x1, y1);
    dzdx = aim_intensity(data->image, x1 + 0.001, y1) - z;
    dzdy = aim_intensity(data->image, x1, y1 + 0.001) - z;
    f_cartesian_to_polar(dzdx, dzdy);
    grad = fmag;
    if (grad == 0.0)
      grad = 0.0001;
    totalweight += length / grad;
    totalheight += z * (length / grad);
}

void measure_edge_contour_difference(graphics_data *data)
{
snake s;
int i, segments;
float *seg;
double gap, pixsize, zoomfac;

    if (data->features->how_close >= 0.0) // if already computed
      return;
    boxes = data->features->boxes;
    hi_x = data->features->hi_x;
    hi_y = data->features->hi_y;
```

```
    lo_x = data->features->lo_x;
    lo_y = data->features->lo_y;
    box_size = data->features->box_size;
    for (i = 0; contour_store[i] > -1.e6 ||
           contour_store[i + 1] > -1.0e6; i += 2) {
    }
    segments = (i / 4) * 4;
    totallength = totalarea = totalsquares = 0.0;
    totalheight = totalweight = 0.0;
    sum_xdx = sum_ydy = sum_dx = sum_dy = sum_x = sum_y = 0;
    sum_wx = sum_wy = sum_x2 = sum_y2 = 0;
    pgf = 0;
    for (i = 0; i < segments; i += 4) {
      seg = &contour_store[i];
      find_nearby_snakes(seg, data);
    }
    pixsize = (data->sem->pixel.x + data->sem->pixel.y) / 2;
    totalarea *= 0.5;
    totalsquares /= 3.0;
    if (totallength == 0.0) {
      printf("Can't match up contours with feature edges.\n");
      data->features->how_close = 999.0;
      return;
    }
    zoomfac = (double) data->curpm->zoom_level / ZOOM_STEPS;
    gap = sqrt(totalsquares / totallength) * zoomfac;
    data->features->how_close = gap * pixsize * 1000;
    gap = totalarea / totallength * zoomfac;
    printf(
      "SEM/AIM difference: RMS = %.3f nM, Average = %.3f nM, Guess:
%.4f\n",
           data->features->how_close, gap * pixsize * 1000,
           totalheight / totalweight);
}
```

---------- Subclaim 1-g -- excerpt from tracker.c -- Feb. 29, 2000 ---
----

Some of the code used in the mathematical algorithm that computes the next iteration's value for the above processing parameter is also used for other purposes and resides in tracker.c and plot.c.

```
double aim_intensity(aimdata image, double X, double Y)
{
int ny, ix, save_nx;
double res;

    if (image == NULL)
      return(0.0);
    save_nx = nx;
    nx = image->nx;
    ny = image->ny;
    ix = (int) ((Y - image->ll.y) / image->del.y) * nx +
          (int) ((X - image->ll.x) / image->del.x);
    if (ix >= 0 && ix < nx * ny - nx - 1)
      res = interpolate(X, Y, image->aimvalues + ix);
    else
```

```
      res = 0.0;
    nx = save_nx;
    return(res);
}

---------- Subclaim 1-g -- excerpt from plot.c -- Feb. 29, 2000 ------
------

static compute_partial_derivatives(aimdata aim)
{
Point_ptr ptr;
int x, y;
int anx, any;
double sx, sy;

    anx = aim->nx;
    any = aim->ny;
    sx = 0.5 / aim->del.x;
    sy = 0.5 / aim->del.y;
    for (x = 0; x < anx; ++x) {
      ptr = aim->aimvalues + x;
      ptr->dzdy = 0;
      for (y = 1; y < any - 1; ++y) {
          ptr += anx;
          ptr->dzdy = (ptr[anx].z - ptr[-anx].z) * sy;
      }
      ptr += anx;
      ptr->dzdy = 0;
    }
    for (y = 0; y < any; ++y) {
      ptr = aim->aimvalues + y * anx;
      ptr->dzdx = 0;
      for (x = 1; x < anx - 1; ++x) {
          ++ ptr;
          ptr->dzdx = (ptr[1].z - ptr[-1].z) * sx;
      }
      ++ ptr;
      ptr->dzdx = 0;
    }
}

double interpolate(x, y, ptr)
double x, y;
Point_ptr ptr;
{
Point_ptr yup;
double delx, dely, fx, fy, yloz, yhiz, z;
double sdlox, sdhix, sdx, sdloy, sdhiy, sdy;
double qx, qy;

    yup = ptr + nx;

    /* Grid size */
    delx = ptr[1].x - ptr->x; /* uM */
    dely = yup->y - ptr->y;

    /* Fraction of the way across the square */
```

```
    fx = (x - ptr->x) / delx; /* dimensionless */
    fy = (y - ptr->y) / dely;

    /* Linear interpolation */
    yloz = ptr->z * (1.0 - fx) + ptr[1].z * fx; /* Joules */
    yhiz = yup->z * (1.0 - fx) + yup[1].z * fx;
    z = yloz * (1.0 - fy) + yhiz * fy;

    /* Quadratic correction */
    sdlox = (ptr[1].dzdx - ptr[0].dzdx) / delx; /* J / uM^2 */
    sdhix = (yup[1].dzdx - yup[0].dzdx) / delx;
    sdx = sdlox * (1.0 - fy) + sdhix * fy;      /* J / uM^2 */
    qx = sdx * ((fx - .5) * (fx - .5) - .25) * delx * delx;

    sdloy = (yup[0].dzdy - ptr[0].dzdy) / dely;
    sdhiy = (yup[1].dzdy - ptr[1].dzdy) / dely;
    sdy = sdloy * (1.0 - fx) + sdhiy * fx;
    qy = sdy * ((fy - .5) * (fy - .5) - .25) * dely * dely;

    z += qx + qy;

    /* Cubic correction - not yet implemented. */

    return(z);
}
```